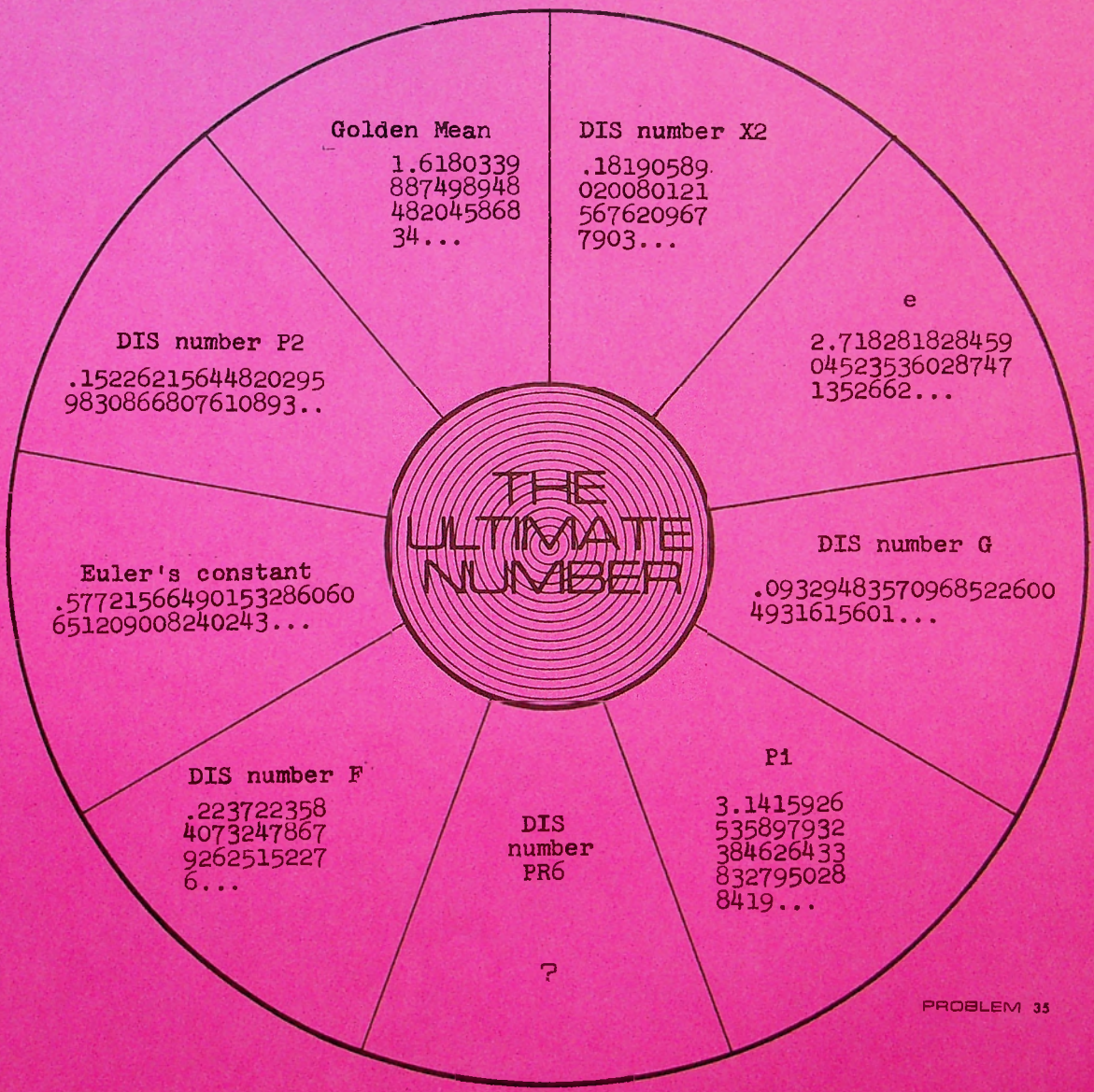
BOX 272
CALABASAS,
CA. 91302

Popular Computing

11

VOL 2 NO 2
FEB 74



THE ULTIMATE NUMBER

Nine numbers are represented on the cover of this issue. Besides the familiar Pi and e, there is the Golden Mean: $(1+\sqrt{5})/2$ and Euler's constant:

$$\lim_{n\to\infty} (1 + 1/2 + 1/3 + 1/4 + \ldots + 1/n - \ln n)$$

plus the DIS numbers P2, X2, F, and G, described in PC8-14. The DIS number PR6 is the DIS sum of prime numbers for which $p\pm6$ is also a prime.

The first eight of these numbers are known to 30 significant digits or more; the number PR6 is not presently known at all. The nine components between them include representatives of the interesting numbers: irrational and transcendental numbers, power functions, the Fibonacci sequence, factorials, and primes.

We seek the product of these nine numbers and make the following offer: a year's subscription to POPULAR COMPUTING for each new significant digit of the Ultimate Number found and verified, up to a maximum of 20. (The limitation is made in case someone is able to calculate the DIS sum for PR6 analytically. We think that it will probably be obtained by actual summation of the primes, and hence that the digits of the Ultimate Number will emerge one by one.) A digit will be considered verified when the subsequent digit is found.

To clarify the DIS process as it applies to the number PR6. For the 1-digit primes, we have only

.05
.07 for a sum of .12

For the 2-digit primes, there are 20 that contribute to the sum:

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $15 per year, or $12 if remittance accompanies the order. For all foreign subscriptions, add $5 per year. Multiple subscriptions to the same address, add $5 each; thus, 3 copies per month is $25 per year, U.S. delivery. Back issues $1.50 each. Subscriptions may begin with any issue. Subscriptions for qualified undergraduate students half price. Copyright 1974 by POPULAR COMPUTING.

Publisher: Fred Gruenberger
Editor: Audrey Gruenberger
Associate editor: David Babcock
Contributing editors: Richard Andree, Paul Armer, Daniel D. McCracken, William C. McGee
Advertising manager: Ken W. Sims
Art director: John G. Scott

@ 2023 This work is licensed under CC BY-NC-SA 4.0

.0011
.0013
.0017
.0019
.....
.....
.0083
.0089
.0097 for a sum of .0972

There are 98 3-digit primes that contribute; 584 4-digit primes; 3660 5-digit primes; and so on. For all the primes of the form p+6 through the prime 9973, the DIS sum is around .297, indicating that the final sum may converge to around .33 or so. Convergence is assured, so the calculation of UN rests on finding PR6.

DESK CALCULATOR REVIEW

Hewlett-Packard HP-45

With the introduction of the HP-45, the price on the HP-35 was dropped to $300. The 45 sells for $400. (Thus the rating for the 35, using the scale given in PC-10, rises from 4.911 to 6.576.) The 45 is the upgraded version of the 35, and hence this review deals only with the added features of the 45 (see the review of the 35 in PC1-6).

All the functions of the 35 are retained, plus the following: factorial (up to 69!); trigonometric functions in degrees, radians, or grads; degrees-minutes-seconds to decimal degrees, and the inverse; 9 addressable words of storage in addition to the 4-level stack storage; metric conversion for volume, weight, and liquid measure; polar to rectangular coordinate conversion and the inverse; percent and change in percent; summation and means and standard deviations. The algorithms used are apparently the same as the corresponding functions on the 35 or 80; for example, sine 4444230 is .4999947727 on both the 35 and 45. The standard deviation formula uses N(N-1) in the denominator.

The rating scale comes out 9.100 for the HP-45, which indicates that the machine offers a great deal of calculating power for its price. A well-written manual comes with the machine, although it is reduced to fit in the machine's box and is thus difficult to read.

ART OF COMPUTING 2

TESTING

When a computer program is first submitted to the machine for assembly or compilation, it is usually in the process of debugging (unless, of course, it contains no bugs, but that condition is rare). If the assembler or compiler does its work well, all the mechanical bugs will be revealed. These include such things as:

1. Undefined symbols.
2. Ambiguous symbols.
3. Mixed mode expressions.
4. Illegal expressions.
5. Impossible paths (that is, statements which cannot be reached in execution).
6. Illegal values.
7. Attempted entry into the middle of loops.

Any of these errors, and others, should inhibit execution and cause the printing of a diagnostic message. When all such bugs have been corrected, a run can be made that will execute, and the program can be said to be the solution to some problem. The question is, is it the problem that was supposed to be solved?

It is at that point that program testing begins. We must verify not only that our program solves the given problem--under controlled conditions--but that it will continue to solve that problem as the input data changes. This task is largely art and is not trivial. It is the task that demands all the ingenuity and low cunning of which the programmer is capable; it is the place where the programmer can fully demonstrate his superiority to the machine.

The two stages--debugging and testing--overlap, since errors in logic are frequently revealed during the debugging stage. Nevertheless, the two stages should be thought of as distinct, since they require radically different techniques. The confusion caused by intermixing the two (which is true of nearly all textbooks that consider either stage) is probably responsible for more bad computing than any other single ill in our field. The trouble is compounded by the unfortunate fact that the concept of program testing is the single most difficult notion to teach in all of computing. All beginners (and many "experienced" programmers) fall into traps like the following:

1. I wrote the program myself, and I have great confidence in me, and so I know it works.

2. I <u>did</u> test my program, and it worked (for one case) and I'm sure it will work for every case.

3. Computers don't make mistakes, and my program prints all its headings correctly, so I know the results are correct.

Any intelligent person is inclined to laugh at such naivete (and admittedly the excuses have been worded as baldly as possible), but what other reasons can there be for the failure to apply even the crudest of test procedures to production programs? Everyone reads of the dramatic glitches: the IRS bill for $.01; the rebate check from the gas company for $999999999; the book club subscriber who receives 3000 copies of the book he rejected--this list is endless. Who tested those programs? Everyone hears about the big, egregious blunders, like the $18 million aborted missile, but the minor lapses in testing simply cause annoyance, followed, sometimes, by efforts at reprogramming. Those who interface daily with program packages wonder sometimes whether anyone tests anything. Or cares.

Two trivial examples will illustrate what this is all about:

(A) A subroutine is to be written to move the words in block A of storage to block B and reverse their order at the same time.

(B) A program is to be written to number all proper fractions that are in lowest terms (a flowchart for this task was shown in issue number 3).

These two tasks call for different testing techniques. In task A, the data can be controlled, and predictable and predicted results can be listed. In task B, there is no data, and the results from a debugged program must be compared to an independent calculation (say, by hand) in order to validate the program.

There is given below a list of guidelines for program testing.

1. You should test the program being tested, and not some other program that you prefer to test. The commonest student error (other than the error of refusing to do any testing at all) is to change the range of the problem so that his work is easier; but then what has he tested? If the real program with the extended range then blows up in his face, who is to blame?

For example, suppose a program is written to generate prime numbers ending in 7 and add their reciprocals. Such a program can only be tested by trying it in a limited range, where the results have been calculated by some other method, and trusting that it will function properly in other ranges.

2. If the program under test admits of data that can be controlled, then the data to be used for the test should be, if possible, all different, and easily generated. All-alike data is always weak for test purposes.

3. The results of a test run should be predictable and be predicted. That is, the tester should know what he expects to see before it is produced. Checking results after they are produced is conducive to believing what one sees, and computer results always look convincing.

4. The human eye cannot be trusted to check more than a few numbers. Any test procedure that calls for printing out block G (25,000 words) to "see if it is now in order" will not work.

5. No program can ever be guaranteed 100%. The goal is to reach a level of confidence at which the person testing can sign off the program and stand behind it (with, perhaps, disclaimers about ranges he has not explored).

6. The test procedure should be logically independent of the program itself. We seek tests that are independent checks, in the same sense that the sum of the interior angles of a regular polygon must be $(N-2)\pi$ radians.

7. If any part of the program's logic can be deleted and the test procedure still works, then it is an inadequate procedure.

8. There must be positive feedback in any test procedure; this means that there must be printed (non-zero) results from the test runs.

To repeat: these are guidelines, not rules. One can readily devise situations for which some of them would have to be relaxed (in particular, the first guideline).

Consider a textbook example. A program is written to count the 1-bits in 1000 words of storage. The program has been debugged; that is, it executes and prints a result. How shall it be tested? It is tempting to load the 1000 words with all-alike numbers, or with checkerboard patterns, but such procedures are weak (although they had better work, to be sure). A more satisfying procedure, in the sense of raising one's confidence level, would be to load the block of storage with 1000 consecutive numbers. Now we have the problem of predicting the number of 1-bits in 1000 consecutive integers. Since the controlled data is systematic, we can expect to find a formula to use to calculate the result. This situation is true more often than not. When the data is under our control for test purposes, systematic data is readily generated and leads to results that can be independently calculated.

Students frequently ask "If the predicted results and the machine results don't agree, how can you tell whether the trouble lies in the program being tested or in the additional code that was written for the test procedure?" You can't. But notice that the converse is much neater: if the test procedure is intelligently designed and the results do agree, then both sets of code have been validated.

Another example: consider the calculation of factorial 10000 (reported in PC4-10). The computer program will produce as output a single number of over 35000 digits. Before committing the computer time to production, the program had to be thoroughly tested. A preliminary run was made to produce (1000!) which was validated by comparison with previous calculations made on a different machine by an independent program (see PC2-10). But that was hardly sufficient; we were proposing to go from a tested result of 2568 digits to one that pushes out into the unknown. So, for this project, independent calculations were made to determine the exact number of digits to be expected; the number of low order zeros; some of the high order digits; and some of the low order non-zero digits. All of these results checked, so that the final result carried a high level of confidence as to its accuracy.

The most important point about program testing is that it should not be a process that begins after debugging. Properly constructed programs should have taken into account the notion "How will I test this program?" during the analysis and flowcharting stages. A tested program is reliable, and "reliability is not an add-on feature." Anyone who sets out to learn computing should be taught the concept of testing from the beginning---from the time when they are first shown how to use a computer to sum the numbers in a block of 1000 words.

Errata

In the definition of the Square Spiral (Problem 24) in PC9-2, the neighbors of square #19 should be squares 6, 5, and 18.

The attendee list for the 14th Symposium (PC9-6) inadvertently omitted the name of Eric Weiss, of Sun Oil Company.

```
10          DIMENSION NUM(7),IVALUE(7),IPREV(7),INPUT(2),IOUT(54)
20          DATA NUM(1),IPREV(1)/2*0/,IVALUE/0,100,50,25,10,5,1/,
30         +  INPUT/4H(I3,,3HI3)/,IOUT/4H(5X,,4HI9HA,4HMOUN,4HT PU,4HRCHA,
40         +     4HSED ,4H$ ,I,4H3/5X,4H,I8H,4HAMOU,4HNT T,4HENDE,4HRED ,
50         +     4H $,I,4H4//1,4H0X,2,4H3HBR,4HOKEN,4H DOW,4HN AS,4H FOL,
60         +     4HLOWS,4H~//1,4H3X,I,4H2,15,4HH DO,4HLLAR,4H BIL,4HL(S),
70         +     4H/13X,4H,I2,,4H12H ,4HHALF,4H DOL,4HLAR/,4H13X,,4HI2,6,
80         +     4HH QU,4HARTE,4HR/13,4HX,I2,4H,8H ,4HDIME,4H(S)/,4H13X,,
90         +     4HI2,7,4HH NI,4HCKEL,4H/13X,4H,I2,,4H11H ,4HPENN,4HY(IE,
100        +     4HS)/)/
110 50      READ(0,INPUT) ITEN, IPUR
120         DO 150 I=2,7
130         IPREV(I)=NUM(I-1)*IVALUE(I-1)+IPREV(I-1)
140 150     NUM(I)=(ITEN-IPUR-IPREV(I))/IVALUE(I)
150         WRITE(1,IOUT) IPUR,ITEN,(NUM(I),I=2,7)
160         GO TO 50
170         END
READY
RUN
200099
     AMOUNT PURCHASED $  99
     AMOUNT TENDERED  $ 200

          BROKEN DOWN AS FOLLOWS-

               1 DOLLAR BILL(S)
               0 HALF DOLLAR
               0 QUARTER
               0 DIME(S)
               0 NICKEL
               1 PENNY(IES)

200091
     AMOUNT PURCHASED $  91
     AMOUNT TENDERED  $ 200

          BROKEN DOWN AS FOLLOWS-

               1 DOLLAR BILL(S)
               0 HALF DOLLAR
               0 QUARTER
               0 DIME(S)
               1 NICKEL
               4 PENNY(IES)

200009
     AMOUNT PURCHASED $   9
     AMOUNT TENDERED  $ 200

          BROKEN DOWN AS FOLLOWS-

               1 DOLLAR BILL(S)
               1 HALF DOLLAR
               1 QUARTER
               1 DIME(S)
               1 NICKEL
               1 PENNY(IES)
```

Speaking of Languages

Robert Teague

In a previous issue (PC7-4) a Fortran programming problem was presented with the challenge that it be coded in fewer than eleven statements. The program was to accept input consisting of the amount purchased by a hypothetical customer and the amount tendered by him in payment for the purchase. The output was to be the change due him, broken down into the minimum number of bills and coins possible.

One reader was able to code the problem in only nine Fortran statements (using proper ANSI Fortran) and is to be congratulated for his ingenuity; he is Clint Erickson, Jr., 23441 Aetna, Woodland Hills, CA 91364, and his solution is given on this page. (Keep in mind that the problem statement ignored run-time efficiency of the code.)

There are two problems this month, one a return to the COBOL program in last month's issue, and the other a new Fortran problem.

The COBOL problem presented in PC9-13 asked three questions concerning the program presented. If the PICTURE clauses for A, B, C, and N were changed to read PICTURE 99, what would your answers to the three questions be?

The Fortran code (a main program and subroutine) presented here was run on a CDC 3170 (the code is machine dependent, but this does not deter from the problem) and produced the line of output shown. Why?

Answers to these problems should be sent to Speaking of Languages, POPULAR COMPUTING, Box 272, Calabasas, CA 91302.

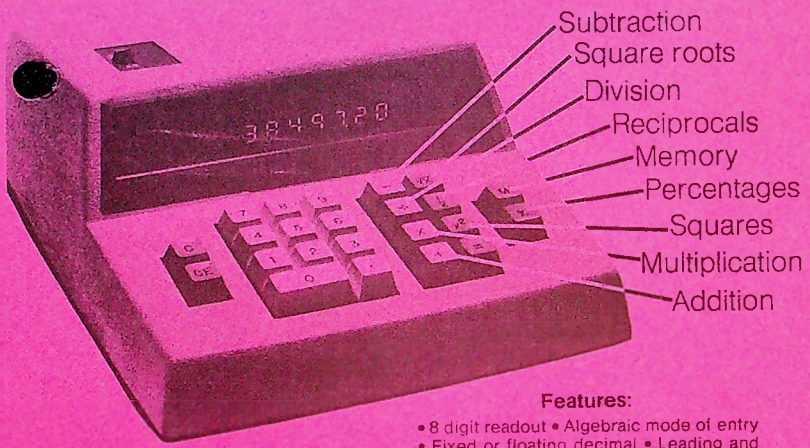
```
20          CALL SUB(3188572,J)
30          L=4HCATS
40          WRITE(1,10) J,L
50  10      FORMAT(/I4,2X,A4)
60          STOP
70          END
80          SUBROUTINE SUB (I,K)
90          I=I+318656
100          K=I/100000
110          RETURN
120          END
READY
RUN

  35   DOGS

STOP 0
```


MITS Presents

The new 908DM, Desk-Top Calculator.



MITS INC.
"Creative Electronics"
6328 Linn, N.E., Albuquerque, New Mexico 87108
505/265-7553 Telex Number 660401

Features:
• 8 digit readout • Algebraic mode of entry • Fixed or floating decimal • Leading and trailing zero suppression • Chain and mixed operations

*Plus the option of programmability.

Full Operation Memory
Memory may be used as:
1. A constant
2. A temporary storage register
3. An accumulator

Indicators:
• True credit balance sign • Overflow

***Prices: 908DM**
Kit. $129.95 Assembled . . $149.95
Size: 8-1/2" x 12" x 3-1/4"

*Programmer

To be used with the MITS 816, 1440, or the new 908DM, desk calculators.
1. Provides 256 programming steps. (With option of expandability to 512 steps.)
2. Stores up to 64 separate programs. Size: 8-1/2" x 12" x 3-1/4"

Instructions:
A. "If Neg." B. "Go To." C. "Return." D. "Remember." E. 2 Run modes of operation

*Programmer Kit . . . . . $199.95 Assembled $299.95
*Combination 908DM and Programmer Kit . . . $299.95 Assembled $399.95

Warranty: Kit: 90 days on parts. Assembled: 2 years on parts and labor.
*Prices subject to change without notice. Available from your local Olson Electronics Dealer

Enclosed is a Check for $ _____
or ☐ BankAmericard # _____
or ☐ Master Charge # _____
Credit Card Expiration Date _____ ☐Kit
Include $5.00 for Postage and Handling ☐Assembled
☐Model 908DM ☐Programmer ☐908DM & Programmer
☐Please Send Information on Entire MITS Line.
NAME _____
ADDRESS _____
CITY _____
STATE & ZIP _____
MITS/ 6328 Linn N.E. Albuquerque, New Mexico 87108 505/265-7553 Telex # 660401
CW-W-74

BOOK REVIEW

A COMPUTER PERSPECTIVE

by the office of Charles & Ray Eames

Harvard University Press, 1973, 174 pages, $15

"A Computer Perspective is an illustrated essay on the origins and first lines of development of the computer.

"The book is based on an exhibition conceived and assembled for International Business Machines Corporation. Like the exhibition, it is not a history in the narrow sense of a chronology of concepts and devices. Yet these pages actually display more true history (in relation to the computer) than many more conventional presentations of the development of science and technology.

"The exhibition...was designed to illuminate the creative forces and social impact of history being made in our own times."

This is the computist's picture book. It begins with a picture of Babbage's punched card, and the schematic he drew of his "Great Calculating Engine." There are photos of every forerunner of the computer, back for nearly 200 years.

Here is Herman Hollerith, his original keypunch and tabulator, and the patent on the latter. The first Russian punched card; early adding machines; a 4-bank set of Triumphator calculators; the Powers line of punched card machines; the first IBM 100 Percent Club; all the people involved with early attempts at automatic computation--this is the scrapbook par excellence. Some of the material goes far afield (Landon and Knox campaign buttons; Tik-Tok from The Road to Oz) but the authors must have been faced with awesome problems of what to include and what to reject. Von Neumann's first flowchart is here, as is a flowchart of Grace Hopper's of 1949.

Although the exhibition was commissioned by IBM, the photos and exhibits do justice to the contributions of Powers, Stibitz, Zuse, Eckert, and Mauchly. The Selective Sequence Electronic Calculator gets two full pages, and the overall view of it shows the central pillars (which were carefully air-brushed out of later photographs).

The history proceeds up to 1950, and includes ENIAC, SEAC, UNIVAC I, the IAS machine, EDSAC, and WHIRLWIND I.

The omissions are interesting. There are no pictures of the 602A, 604, or the CPC, each of which was a landmark. Software is neglected, except for a reference to COBOL. Patents are shown, but not the patent on the rectangular hole.

After browsing through the book, the reader should then browse through its index, to discover more hidden treasures. Listings like "Elliot Frog and Switch Company" in the index are bound to trigger more browsing. The book is an almost endless delight.

The Eames group has gone to extraordinary pains to obtain the original documents and facts about our history. This is surely a book that can be recommended as belonging on the bookshelf of anyone interested in computing.

--FJG

BIT COUNTING

PROBLEM 36

Given a word-oriented binary computer with 32768 words of core storage. A program is to be written that will reside at the top end of core, from word S through word 32767. This program will consist of two loops. The first loop will load every word of core from 00000 through S-2 with its own address; that is, word 00001 is to contain the number 1; word 00002 is to contain the number 2; and so on. The second loop counts the number of 1-bits in all 32768 words of core storage and puts that count in word S-1.

Problem: To predict the final contents of word S-1.

The conditions of this problem will vary from machine to machine, and with the length of the program. The following formula:

$$\sum_{i=0}^{J-1} 2^i \left[\frac{M}{2^{i+1}} \right] + \mathrm{MAX}\left[M\left(\mathrm{mod}\ 2^{i+1}\right) - 2^i + 1,\ 0 \right]$$

gives the number of 1-bits in the integers from 1 to M. J is the "width" of M in bits. For example,

$$32700)_{10} = 111111110111100)_2 \quad \text{and } J = 15.$$

The brackets denote "greatest integer in." For the number 32700, the formula yields 244952.

The following program will evaluate the bit-count formula for the integers from X to Y.

```
100 DEF FNA(X,Y)=FNB(Y)-FNB(X-1)
110 DEF FNB(M)
120 IF M>0 THEN 150
130 FNB=0
140 GØTØ 250
150 J=INT(LØG(M)/LØG(2))+1
160 T=0
170 FOR I=0 TØ J-1
180 T1=2↑I*INT(M/2↑(I+1))
190 T2=FNC(M,2↑(I+1))-2↑I+1
200 IF T2>0 THEN 220
210 T2=0
220 T=T+T1+T2
230 NEXT I
240 FNB=T
250 ENDFNB
260 DEF FNC(X,Y)=X-Y*INT(X/Y)
```


? FRUSTRATED

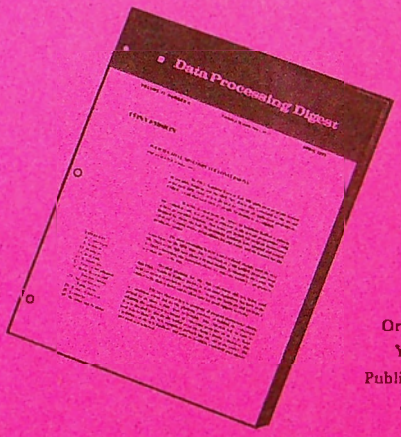
TRYING TO FIND *GOOD* COMPUTER LITERATURE. . .

. . . AND THE TIME TO READ IT?

Hire full-time research for just $4.25 a month! Here's what you get:

1. a staff of computer pros continuously monitoring the computer literature
2. a technical library source of 59 computer publications and 125 trade/management publications
3. news of conferences, meetings, seminars
4. reviews of new books
5. original reports about problems faced and solved, but not yet reported in the literature

. . . presented in report form each month. Write for information about DATA PROCESSING DIGEST. Or send $4.25 for our current issue and apply to your continuing subscription (12 issues, $51).



Abstracts
Digests
Resources
News Items
Calendar
Reviews
Original Reports
Yearly Index
Published Each Month
Since 1955

Name _____
Dept. _____
Company _____
Address _____
City _____ State _____ Zip _____
PC

Data Processing Digest, Inc.
6820 LA TIJERA BOULEVARD, LOS ANGELES, CALIFORNIA 90045 / PHONE (213) 776-4334

N-SERIES

Log 11	1.041392685158225040750199971243024241706702190466 45
Ln 11	2.397895272798370544061943577965129299821706853937 42
$\sqrt{11}$	3.316624790355399849114932736670686683927088545589 35
$\sqrt[3]{11}$	2.223980090569315521165363376722157196518699128096 92
$\sqrt[5]{11}$	1.615394266202178001508147882063835845416535205469 29
$\sqrt[7]{11}$	1.408543888428699411406584628756831160498853412347 82
$\sqrt[10]{11}$	1.270981615210140638605535137528442423923879156374 91
$\sqrt[100]{11}$	1.024268759600549626327283083876020346649268178229 34
e^{11}	59874.141715197818455326485792257781614261079695741 0
π^{11}	294204.017973890597105695642007184667027564721139603
$\tan^{-1}$ 11	1.480136439594151505732935016331285824072531803371 00
11^{1000}	24699329180058263341240883850852214777097333852383962348691829518307393903754331753678661164569461919738035611890365233633533798726571008961243792655536655282201820357872673322901148243453211756020067624545609411212063417307681204817377763465511222635167942816318177424600927358163388910854695041070577642045540560963004207926938348086979035423732739933235077042750354729095729602516751896320598857608367865475244863114521391548985943858154775884418927768284663678512441565517194156946312753546771163991252528017732162399536497445066348868438762510366191040118080751580689254476068034620047646422315123643119627205531371694188794408120267120500325775293645416335230014278578281272863450085145349124727476223298887655183167465713337723258182649072572861625150703747030550736347589416285606367521524529665763903537989935510874657420361426804068643262800901916285076966174176854351055183740078763891951775452021781225066361670593917001215032839838911476044840388663443684517735022039957481918726697789827894303408292584258328090724141496484460001

EXTENDED ARITHMETIC

Given a calculating device that can operate on 8-digit operands to produce an 8-digit result. Addition and subtraction of larger numbers can be done by parts fairly obviously:

3.1415926	53589793
+2.7182818	28459045
5.8598744	82048838

Multiplication of 8-digit numbers to produce a complete 16-digit product is also quite straightforward:

		A 3141	5926 B
		C 2718	2818 D
		1669	9468
	0885	1338	
	1610	6868	
0853	7238		
853	9733	9875	9468

If the four parts of the original 8-digit numbers are labelled as shown, then the partial products are, in order, BD, AD, CB, and AC, and each of these is 8 digits long, leading to the full 16 digit product.

Extending a quotient can be done as follows. Suppose we wish to obtain 16 digits of the quotient of the numbers

$$\frac{3.1415926}{2.7182818}$$

An 8-digit machine will readily produce the first 8 digits of the quotient: 1.1557273. This quotient can be multiplied back by the divisor, using the previously explained methods of double precision multiplication. The product in this case is 3.14159248535314. If this is subtracted from the dividend, the remainder is 11464686, and that remainder, divided again by 2.7182818, gives the next 8 digits of the quotient, 42176222.

All of which leads to a method of extending a divisor. Label the parts of the division as follows:

$$\frac{3.1415926\ \ 53589793}{2.7182818\ \ 28459045} = \frac{A + B}{C + D}$$

$$\frac{A + B}{C + D} = \frac{A + B}{C\left(1 + \frac{D}{C}\right)} = \frac{A + B}{C}\left(1 - \frac{D}{C} + \frac{D^2}{C^2} - \cdots\right)$$

$$\approx \frac{A}{C} + \frac{B}{C} - \frac{AD}{C^2}$$

The calculation of A/C must be carried to 16 places, as outlined above. The other two terms need only be carried to 8 digits. The final calculation, then, is:

```
1.1557273  42176222
         + 19714583
         - 12099884
-------------------
1.1557273  49790921
```

which is correct to 16S.

In the early days of computers, when the operation code DIVIDE was a costly frill, special methods were devised to yield a quotient by iterative schemes. For example, the Newton-Raphson scheme can be applied to the equation $1/x - N = 0$, leading to the recursion:

$$x_{n+1} = x_n(2 - Nx_n)$$

where N is a number whose reciprocal is desired. If expressed in the form

$$x_{n+1} = x_n + x_n(1 - Nx_n)$$

then the only double precision calculation required is in the term Nx_n.

For example, to obtain the reciprocal of N = 7, using a starting value of $x_0 = .1$:

$$x_1 = .1 + .1(1 - .7) = .13$$

$$x_2 = .13 + .13(1 - .91) = .1417$$

$$x_3 = .1417 + .1417(1 - .9919) = .14284777$$

As a method for extending a divisor on a pocket calculator, this would be a slow and very tedious process.

pC11-16

ANNUAL INDEX

Volume 1, 1973 April-December
Numbers 1-9. 144 pages